Master en investigación

Curso 2007-2008

FAFNER-2:

Adaptación al Grid de un código para estimar el calentamiento de los plasmas de fusión

Manuel Rodríguez Pascual

Dirigido por:

Dr. Rafael Mayo García
Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas
Facultad de Ciencias Físicas, Universidad Complutense de Madrid
Dr. Ignacio Martín Llorente
Facultad de Informática, Universidad Complutense de Madrid

Facultad de Informática
Universidad Complutense de Madrid

I

"And Lord, we are especially thankful for nuclear power, the cleanest, safest energy source there is. Except for solar, which is just a pipe dream"

Homer Simpson

Resumen

FAFNER-2 es un código de modelización para la inyección de partículas neutras rápidas en un plasma toroidal tridimensional, destinado a calcular la trayectoria de los iones rápidos resultantes hasta que se pierden en el sistema o se incorporan al plasma. En este trabajo se detallará el proceso de gridificación de esta aplicación. Se explicarán los pasos seguidos para transformarla, pasando de utilizar paso de mensajes con SHMEM en una arquitectura MIPS, a utilizar el Grid corriendo sobre X86. Así mismo, se evaluarán las mejoras de rendimiento y portabilidad obtenidas en cada fase del proceso.

Abstract

FAFNER-2 is a code which models the injection of fast neutral particles into 3-dimensional toroidal plasmas, employed to calculate the trajectories of the resulting fast ions until they are either lost or fully thermalised. In this work, the gridification proccess of this application will be described. The steps of the transformation from a SHMEM over MIPS to a Grid over X86 application will be fully explained. Also, performance and portability gains obtained on each step of the proccess will be evaluated.

Agradecimientos

Este trabajo parte de una propuesta de los Doctores Francisco Castejón y José Guasp.

En primer lugar quiero agradecer al Dr. Francisco Castejón sus explicaciones y gráficos sobre el funcionamiento de los reactores de fusión, y en concreto sobre la técnica NBI de calentamiento del plasma.

Tanto el código fuente de FAFNER-2 como los datos de muestra que se emplearon como ejemplo de ejecución me han sido facilitados por el Dr. José Guasp. Así mismo, sus consejos y sugerencias fueron una gran orientación a la hora de entender y modificar el código de la aplicación.

Quiero agradecer la ayuda de Antonio Muñoz Roldán y Angelines Alberto Morillas, de la Unidad de Supercomputación y Desarrollo Grid del CIEMAT, a la hora de lidiar con las diferentes arquitecturas hardware y sus particularidades. Sin sus consejos y experiencia en el *application porting*, este trabajo no habría sido posible.

Javier Fontán, del grupo de Arquitectura de Sistemas Distribuídos de la Facultad de Informática UCM supuso una ayuda a la hora de dominar el uso de GridWay, especialmente con trabajos complejos.

Y por supuesto, me gustaría agradecer a Antonio Juan Rubio-Montero, de la Unidad de Arquitectura Informática del CIEMAT, su ayuda y apoyo a lo largo de estos meses, especialmente en los momentos más difíciles.

En cualquier caso, este trabajo sólo ha sido posible gracias al trabajo diario del personal dedicado al mantenimiento y mejora de las instalaciones del Centro de Cálculo de Madrid del CIEMAT, donde se albergan la mayoría de los equipos utilizados. Para ellos, y especialmente para el Director de la División de TIC Fernando Blanco, mi más sincero agradecimiento.

Índice general

Índice de figuras

Capítulo 1

Introducción

La fusión nuclear aspira a ser la fuente de energía del futuro. Utilizando como combustible el deuterio y tritio (dos isótopos del hidrógeno presentes en el agua de mar), y sometiéndolos a condiciones extremas de temperatura y presión, se consigue producir la misma reacción que ocurre en el interior de las estrellas, produciendo una cantidad enorme de energía. Y lo mejor es que es un combustible prácticamente inagotable: un kilómetro cúbico de agua de mar produciría la misma energía que todas las reservas de petróleo conocidas.

El principal problema de la fusión nuclear es de tipo tecnológico: todavía no somos capaces de crear un reactor que pueda crear una reacción de forma sostenida en el tiempo, y que además produzca más energía de la que se le suministra. Por el contrario, los reactores actuales necesitan cantidades inmensas de electricidad para funcionar, por lo que los experimentos son muy costosos.

Para solucionar este hecho, se recorre a la simulación por ordenador. Mediante el uso de herramientas software, es posible analizar el comportamiento del reactor, ajustar los distintos parámetros de funcionamiento, y estudiar qué ocurre. De ese modo, por una fracción del precio que supondría llevar el experimento a cabo, es posible calcular cuál sería el resultado del mismo. El inconveniente es que se necesita una gran capacidad de cálculo para poder realizar estas simulaciones en un periodo razonable de tiempo.

La tecnología Grid, por su parte, posibilita la compartición de recursos, tales como tiempo de proceso y almacenamiento, entre diferentes entidades. Empleando esta tecnología, en pocos años se ha conseguido crear una enorme infraestructura, donde los equipos de cientos de cen-

tros de investigación de decenas de países trabajan juntos para resolver problemas de gran magnitud, en uno de los ejemplos más destacados de colaboración a nivel mundial.

Así pues, la Grid y la simulación de fusión nuclear se complementan perfectamente, ya que uno proporciona las herramientas necesarias para resolver los problemas que el otro plantea. Y este es precisamente el objetivo de este trabajo: lograr que un simulador de partículas pueda ejecutarse en la Grid, permitiéndonos estudiar el comportamiento de una parte de un reactor de fusión.

Capítulo 2

FAFNER-2

2.1. Introducción a la Fusión Nuclear

La fusión nuclear es el proceso mediante el cual dos núcleos atómicos se unen para formar otro mayor. El nuevo núcleo tiene una masa inferior a la suma de las masas de los dos núcleos que se han fusionado para formarlo, y esta diferencia de masa es liberada en forma de energía siguiendo la relacción $E = mc^2$.

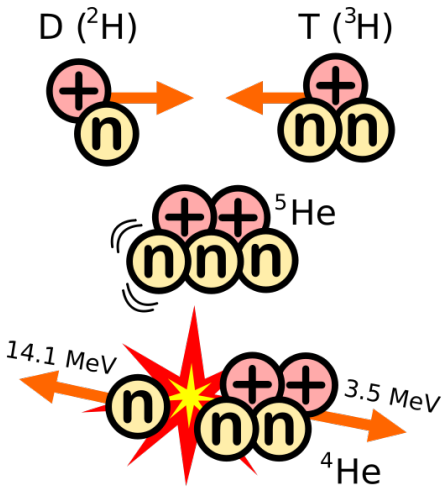


Figura 2.1: Reacción base de la fusión nuclear

La reacción de fusión más sencilla es la que une el deuterio y el tritio formando helio y liberando un neutrón, según la ecuación

$$^2H + ^3H \rightarrow ^4H + n. \tag{2.1}$$

La figura 2.1 representa gráficamente este proceso.

Los núcleos atómicos tienden a repelerse debido a que tienen carga positiva, de modo que cuanto más cerca están, mayor es la fuerza repulsiva. Sin embargo existe otro proceso, las fuerzas nucleares atractivas -o interacción fuerte-, que son muy fuertes a pequeñas distancias. Eso hace que la fusión nuclear sólo pueda darse en condiciones de enorme Temperatura (del orden de 300 millones de grados centígrados) y Presión. En ese momento, las fuerzas nucleares atractivas superan a las de repulsión y los átomos comienzan a fusionarse.

En los reactores de fusión, esta Temperatura y Presión se obtienen confinando el material a fusionar en un Campo Magnético, mientras se va aumentando su Temperatura y Presión. Bajo estas condiciones, el hidrógeno alcanza el estado de plasma.

2.2. El calentamiento por NBI

Figura 2.2: Inyector de partículas neutras del TJ-II del CIEMAT

El calentamiento por inyección de átomos neutros, o NBI por sus

siglas en inglés (*Neutral Beam Injection*) es una técnica para calentar el plasma mediante la inyección de átomos neutros de alta energía.

Estos átomos, al ser neutros, pueden atravesar sin problemas el campo magnético en el que está confinado el plasma. Ahí colisionan con las partículas existentes (iones, electrones y átomos fríos), depositando así su energía, y pasando ellas mismas a formar parte del plasma.

La fotografía 2.2 muestra uno de los tres inyectores del TJ-II del CIEMAT. El esquema 2.3 detalla sus partes de un modo gráfico. Su funcionamiento es el siguiente:

1. *Ion Source*. Se genera un plasma frío.
2. Los iones de este plasma se aceleran hasta alcanzar una gran Energía, de aproximadamente 40 KeV.
3. *Neutralizer*. Dado que queremos inyectar partículas neutras y no iones, en esta etapa a los iones se les deja sin carga.
4. *Ion Beam Deflection Magnet*. En esta etapa los iones que aún tienen carga son deflectados mediante el uso de un Campo Magnético, que atrae a las partículas cargadas y las elimina.
5. *Calorimeter*. Esta es una etapa utilizada para comprobar el correcto funcionamiento del inyector. Se coloca un calorímetro que mide la Energía de las partículas neutras. Cuando se comprueba que todo es correcto, el calorímetro es retirado para que dichas partículas puedan entrar al reactor.

Los problemas de esta técnica son dos. Por un lado, el hecho de que siempre que queramos calentar el plasma provoquemos que éste aumente su densidad y pueda llegar a producir un colapso. Y dado que esta es una técnica muy costosa, el uso de simuladores es primordial. Esa es precisamente la función de FAFNER-2, simular los eventos físicos que ocurren cuando las partículas neutras entran en el plasma.

2.3. Descripción de FAFNER-2

FAFNER-2 es un código diseñado para la modelización de inyección de partículas neutras rápidas en plasmas toroidales tridimensionales, y calcula la trayectoria de los iones rápidos resultantes hasta que se pierden en el sistema o son totalmente absorbidos por el plasma.

El código original de FAFNER fue publicado por GG. Lister, del instituto Max-Planck, en 1985 [8]. FAFNER-2, una revisión de este software, se publicó un año después. La versión de la que se hizo este trabajo es una modificación de FAFNER para adaptar su funcionamiento, ecuaciones y variables a la geometría del reactor TJ-II del CIEMAT. Esta versión fue creada por el departamento de fusión del CIEMAT en

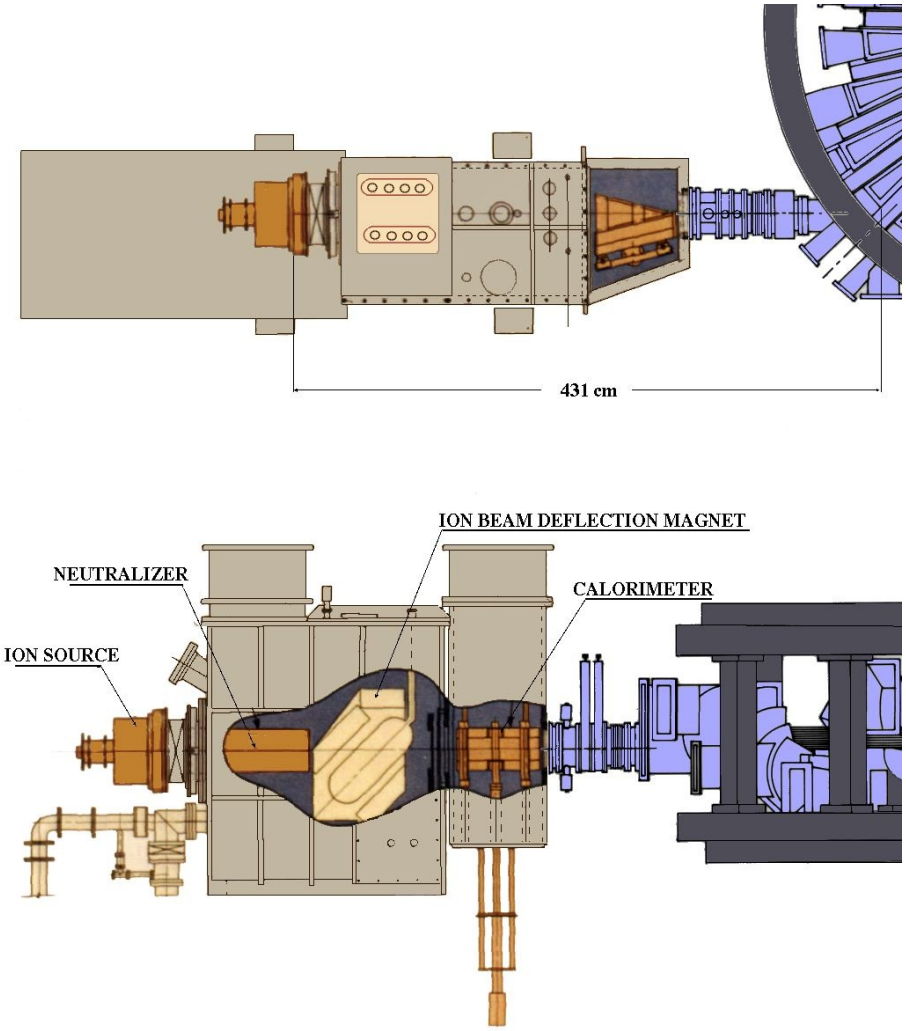


Figura 2.3: Esquema de un inyector de partículas neutras

colaboración con los autores originales [10].

La primera parte del código se encarga de simular la inyección de partículas neutras en el plasma. Se obtiene como resultado las coordenadas de dichas partículas y su velocidad. A continuación, en la segunda parte del código esta información es usada para determinar la ionización inicial de los átomos neutros en el plasma, y después, la distribución de energía al plasma como resultado de su interacción con los iones rápidos. También se computan en detalle las pérdidas de energía como resultado de las colisiones de las partículas neutras emitidas por cada fuente con los conductos y aperturas.

Desde el punto de vista matemático, se utilizan técnicas de Monte Carlo para computar las coordenadas de un conjunto de iones rápidos, correspondientes a las ráfagas de partículas neutras y los parámetros del plasma apropiados para la modelización. Las trayectorias que siguen esos iones, y la consiguiente interación con el plasma son descritos en términos de una equación de Fokkner-Plank. Las trayectorias de los iones rápidos son resueltos en coordenadas de flujo de 'Boozer'. Las técnicas numéricas empleadas son bien conocidas, y han sido empleadas previamente en describir el calentamiento de los haces neutros tanto en tokamaks como en stellarators.

El código se ha construído con la intención de que sea adaptable a un código de transporte o equilibrio. Este aspecto será descrito más profundamente en el capítulo 6.

2.4. Configuración del haz de partículas neutras

2.4.1. Geometría

La configuración del haz de partículas neutras empleada en FAFNER-2 está ilustrada en las figuras 2.4, 2.5 y 2.6. Se han adoptado dos sistemas diferentes de coordenadas:

1. La geometría del plasma y el toroide son descritas utilizando un sistema de coordenadas cilíndricas *(R, ϕ, z)* con respecto al centro O del toroide que contiene el plasma, donde el ángulo toroidal ϕ es medido con respecto a la línea OC (normalmente una de las espirales del campo toroidal). El punto de referencia en el toroide para el haz n es el centro del conducto de inyección D, localizado en *(R_D, ϕ_D)*.

2. La geometría de la fuente de partículas neutras de cada haz (fig 2.4) está descrita usando coordenadas cartesianas *(x,y,z)*, con un origen S a una distancia R_s de O a lo largo de la proyección de la línea OD. El eje x está definido en el plano medio del toroide, con un ángulo θ_I a OS.

El haz central está dirigido con un ángulo β sobre la horizontal (figura 2.5) en el plano X-Z, y de tal modo que su proyección en el plano horizontal intersecta el eje SQ con un ángulo α (figura 2.4) en el plano X-Y.

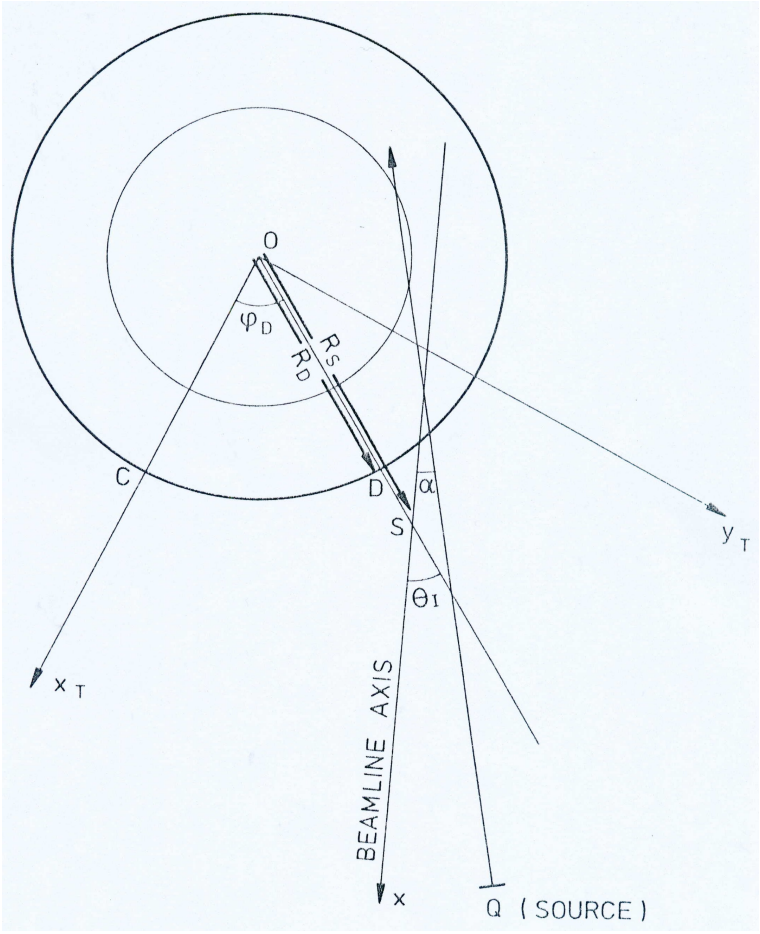


Figura 2.4: Sistema de coordenadas del haz: plano medio del toroide

2.4.2. Fuente de iones y *Neutraliser*

El código proporciona una opción para detallar la fuente de iones. En el caso que se vaya a inyectar un isótopo de hidrógeno, se emitirán tres especies de iones, H^+, H_2^+, y H_3^+. Estos iones poseen una fracción de potencia relativa p_1^+, p_2^+, y p_3^+ respectivamente. Así, la potencia máxima disponible de la fuente de partículas neutras, contando con la pérdida

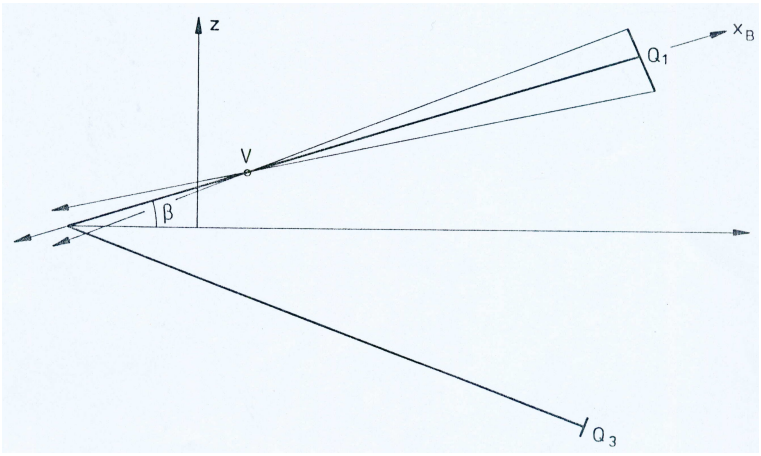


Figura 2.5: Vista lateral del haz de partículas

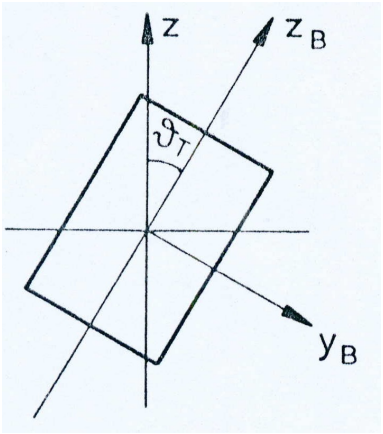


Figura 2.6: Inclinación de los inyectores

de rendimiento durante la fase de neutralización, es

$$P_N(max) = \sum_k P_S p_k^+ n_k \tag{2.2}$$

donde k va de 1 a 3 (para los tres tipos de iones) y asumimos la normalización

$$p_1^+ + p_2^+ + p_3^+ = 0. \tag{2.3}$$

Sin embargo hay cuatro razones por las que se puede producir una pérdida de la potencia disponible:

1. Ineficiencia de la fuente de iones.
2. Mecanismo de neutralización no ideal.
3. Re-ionización entre el neutralizador y el plasma.
4. Colisiones de las partículas neutras con los conductos y aperturas.

El punto 4 es tratado en profundidad en la sección 2.4.3, mientras que los puntos 1 a 3 son simulados colectivamente mediante un factor de eficiencia ϵ_k. Por tanto, la potencia suministrada por cada fuente de partículas neutras es:

$$P_N = P_S \sum_k P_S p_k^+ n_k \epsilon_k \tag{2.4}$$

La fracción de energía de cada tipo de partículas en el haz de partículas neutras es

$$p_k^n = \frac{k p_k}{\sum_k k p_k} \tag{2.5}$$

Este parámetro es usado en el código con fines estadísticos.

2.4.3. Conductos y Aperturas

Habitualmente es necesario dar al haz una forma determinada antes de que entre en el toroide. Para ello se le hace pasar a través de una superficie sólida en la que se han practicado aperturas con una forma concreta. Un determinado número de partículas colisionarán contra el sólido, quedando el resto con la forma deseada.

La fracción f_L de partículas perdidas en este apartado es tenida en cuenta por FAFNER-2, pudiendo elegir entre diferentes tipos y tamaños de aperturas. Con estos datos puede calcularse la potencia total introducida en el toroide, que es

$$P_I = (1 - f_L)P_N \tag{2.6}$$

2.5. Descripción del plasma

El modelo del plasma consiste en un total de n_p especies diferentes de ion. Asumimos que el plasma está confinado en un toroide de radio r_L, cuyo centro está en R_V.

Las superficies de flujo están definidas por una coordenada radial efectiva s=$\sqrt{\psi/\psi_e}$, donde $2\pi\psi$ es el flujo toroidal y ψ_e es el valor de ψ en el borde del plasma.

Para definir los parámetros del plasma se utiliza un conjunto discreto de n_F puntos, s_J, y cada superficie de flujo j abarca un volumen V_j. El volumen encerrado por una superficie de flujo ψ es definido por una serie de coeficientes $B_V^{(v)}$ tales que

$$\tilde{V}(\tilde{\psi}) = \sum_{v=1}^{(v)} \tilde{\psi}^v. \tag{2.7}$$

El volumen contenido por $s(n_F)$ es el volumen del plasma V_P.

2.5.1. Tratamiento de las impurezas

El número de impurezas puede ser introducido en FAFNER-2 en forma de tablas, que deben ser obtenidas de resultados experimentales.

En ausencia de dicha información, se considera que la carga efectiva del plasma es constante.

2.5.2. Densidad de partículas neutras en el plasma

La densidad de átomos neutros en el plasma contribuirá a la pérdida de energía disponible, ya que los iones rápidos pueden ser re-neutralizados en un intercambio de carga, y por tanto perderse. La densidad de partículas neutras, n_O, es modelada en el código según la ecuación

$$n_O(s) = n_O(r_L)10^{-\alpha_O(1-s)} \tag{2.8}$$

donde $n_O(r_L)$ es la densidad en el radio máximo y α_O es un coeficiente apropiado.

2.5.3. Magnitud del Campo Magnético

El campo magnético tridimensional utilizado en los cómputos se define en términos de las coordenadas de Boozer (ψ, ϕ, θ). Su magnitud se define como

$$B = \sum_{m,n} A_{mn}(\tilde{\psi})cos(n\phi - m\theta), \tag{2.9}$$

donde $\tilde{\psi} = \psi\psi_e$. Los coeficientes A_{mn} se definen por los polinomios

$$A_{mn}(\tilde{\psi}) = \sum_{v=0}^{3} A_{mn}^{(\tilde{\psi})}\tilde{\psi}v + \frac{1}{2}l(m,n) \tag{2.10}$$

Los valores de *m,n* y *l(m,n)*, junto con ψ_e y conjuntos de constantes $A_{mn}^{(\tilde{\psi})}$, $B_g^{(v)}$, $B_I^{(v)}$, $B_i^{(v)}$ son leídos de un fichero de datos. Estos coeficientes han sido calculados asumiendo que el Campo Magnético medio en el eje magnético es de un Tesla. El programa incorpora rutinas para ajustarlos según el valor real del campo.

2.5.4. Campos Eléctricos

Los campos eléctricos radiales se definen en términos del potencial eléctrico $\Phi(\psi)$ de la siguiente manera:

$$\Phi(\tilde{\psi}) = \Phi(0)(1 - \tilde{\psi}^{m_\Phi})^{n_\phi} \tag{2.11}$$

2.6. Descripción del modelo físico

2.6.1. Deposición de los iones rápidos (Cálculo de (H(R))

La deposición de iones rápidos en el plasma debido a la interacción con las ráfagas de partículas neutras está gobernada por la ecuación

$$F(s) = \int f(\overrightarrow{x}_i, \overrightarrow{v}_i) e^{-\int_0^x dl/\lambda(\overrightarrow{x}_p, \overrightarrow{v}_i)} d\overrightarrow{x}_i, d\overrightarrow{v}_i \tag{2.12}$$

donde *F(s)* representa el flujo de partículas neutras cruzando la superficie del flujo s, $f(\overrightarrow{x}_i, \overrightarrow{v}_i)$ es la función que modela la fuente de partículas neutras con coordenadas de espacio y velocidad $\overrightarrow{x}_i$ y $\overrightarrow{v}_i$ respectivamente, y λ es la media del camino libre hacia las colisiones para todos los procesos de ionización y carga-descarga, que es una función de $\overrightarrow{v}_i$ y de las coordenadas del plasma $\overrightarrow{x}_p$. La integral de dl representa una integración a lo largo del camino de cada partícula neutra desde su punto de origen a su intersección con la superficie del flujo s, a una distancia x.

La media del camino libre hacia las colisiones está definida por

$$\lambda = v/v_B \tag{2.13}$$

donde

$$v_B = < \sigma v >_e n_e + \sum_k (\sigma_i^k + \sigma_{cx}^k) v n_i^k. \tag{2.14}$$

En esta ecuación, $< \sigma v >_e$ es el coeficiente de las ionizaciones por impactos entre electrones y partículas neutras entrantes, v es su velocidad, n_e es la densidad local de electrones, n_i^k la densidad de especies

de ion k y σ_i^k y σ_{cx}^k son las secciones eficaces para los impactos de iones e intercambios de carga entre los iones del plasma y las partículas neutras inyectadas. En esta versión del código, se asume que el plasma está compuesto por distintos isótopos de hidrógeno (con impurezas), y las partículas inyectadas son isótopos de hidrógeno u oxígeno.

Las secciones transversales para el impacto de iones en un plasma de hidrógeno son:

$$\sigma_i = \sum_{m=0}^{6} C_m (ln(E/A))^m \tag{2.15}$$

donde E/A es la Energía por Nucleo y C_m son un conjunto de coeficientes predefinidos.

Las seciones transversales para el impacto con iones de impureza son:

$$\sigma_I = 4{,}6 \cdot 10^{-16} Z_I (1 - e^{-\Sigma)}/\Sigma \tag{2.16}$$

en donde Σ se puede tomar como

$$\Sigma = \frac{E}{3{,}2 \cdot 10^4 A_B Z_I} \tag{2.17}$$

en el caso de que E/A_B sea mayor de 50KeV por núcleo, siendo A_B el número de iones inyectados y Z_I el número de carga de los iones de impureza del plasma.

En el caso contrario, si E/A_B es menor de 50keV se emplea el límite

$$\sigma_I = 4{,}6 \cdot 10^{-16} Z_I \tag{2.18}$$

Las secciones transversales para el intercambio de carga sólo están disponibles en el código si se inyecta hidrógeno, deuterio o tritio en un plasma de isótopos de hidrógeno. En ese caso,

$$\sigma_{cx} = \frac{6{,}937 \cdot 10^{-15}(1 - 0{,}115(log_{10}\tilde{\Sigma})^2}{(1 + 1{,}112 \cdot 10^{-15}\tilde{\Sigma}^{3,3})} \tag{2.19}$$

donde $\tilde{\Sigma} = E/A_B$.

Las partículas neutras que escapan del plasma y colisionan con las paredes del toroide suponen una importante contribución a las impurezas del plasma debido al bombardeo iónico. El código de FAFNER-2 sólo permite calcular este bombardeo en el caso de que las paredes del toroide sean de hierro o molibdeno, aunque introducir las constantes necesarias para calcular este valor en el caso de otros metales es una tarea trivial. La ecuación que determina el bombardeo iónico es

$$I_S^O = \sum_j \sum_k f_s(j,k) C_s(K, N_I, N_W) I_I(j) \tag{2.20}$$

donde $f_s(j,k)$ es la fracción de potencia de la fuente j, con energía k que viaja por el plasma sin ionizar, $C_s(K, N_I, N_W)$ es el coeficiente de bombardeo e I_I es el actual de la fuente j.

2.6.2. Deposiciones de Energía

La interacción de los iones rápidos con el plasma de fondo, $f(v, \zeta, t)$ en el plasma obedece a

$$\frac{df(v,\zeta,t)}{dt} = \frac{1}{2t_E v^2} \frac{\delta}{\delta v} \{(v^3 + v_c^3) f + \frac{v}{2E_b}(v^3 kT_e + v_c^3 KT_i) \frac{\delta f}{\delta v}\} + v_a \frac{\delta}{\delta \zeta}(1-\zeta^2) \frac{\delta f}{delta\zeta} \tag{2.21}$$

donde t_E es el tiempo de intercambio de energía de Spitzer y v_c es la velocidad crítica.

Los iones rápidos también pueden cambiar carga con las partículas neutras del plasma, y por tanto perderse. La sección transversal para este proceso en el caso de inyección de hidrógeno o deuterio es la de la ecuación. Las partículas neutras rápidas resultantes pueden ser reionizadas en otra parte del código, pero su efecto no está incluído en esta versión del código.

2.7. Programación de FAFNER-2

FAFNER-2 está programado empleado Fortran90. Como mecanismo de comunicación entre procesos usa SHMEM, que a lo largo de este trabajo será substituido por MPI 2.

Además, implementa dos librerías con funciones auxiliares, lib_g3d y libfa2_sta. La primera contiene funciones necesarias para resolver determinadas ecuaciones relaccionadas con la geometría 3D del sistema. La segunda, para resolver las ecuaciones explicadas en los puntos anteriores (2.4 a 2.6), y estudiar la evolución de las partículas a lo largo del tiempo. La librería libfa2_sta también emplea SHMEM, por lo que ha sido necesario adaptar su código.

2.8. Resultados

La información que proporciona una ejecución de FAFNER-2 es doble.

Por un lado, presenta la posición y velocidad final de un determinado número de partículas. Esto puede representarse con un programa externo de manera gráfica, obteniendo un resultado similar a 2.7.

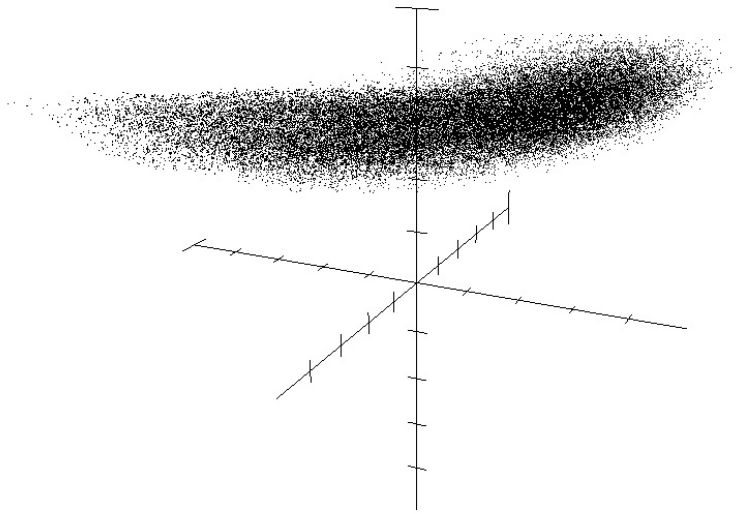


Figura 2.7: Representación 3D de los resultados de una ejecución típica de FAFNER-2

Además, presenta estadísticas varias sobre el estado del plasma. Un ejemplo se puede ver en la figura 2.8. La gráfica muestra el porcentaje de partículas en las que ocurrió un intercambio de carga, la potencia

absoluta suministrada al toroide, y el porcentaje de partículas que se perdieron.

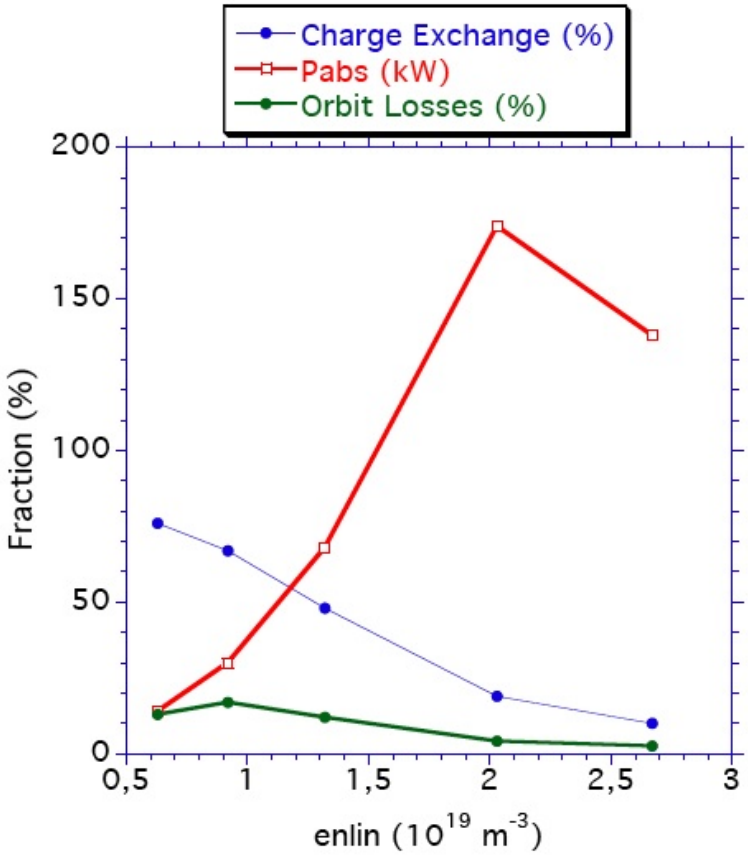


Figura 2.8: Estadísticas sobre el estado del plasma proporcionadas por FAFNER-2

Capítulo 3

Conceptos Relacionados

La finalidad de este trabajo es modificar FAFNER-2 para adaptarlo a dos paradigmas de procesamiento paralelo (paso de mensajes y Grid). Por tanto, para entenderlo es fundamental tener una amplia visión de estas tecnologías.

3.1. Procesamiento Paralelo

Podemos definir procesamiento paralelo como el método que engloba diferentes técnicas empleadas para proporcionar múltiples ejecuciones simultáneas de un código de programación -o parte del mismo- con el fin de mejorar su velocidad de ejecución. Estas técnicas se emplean a varios niveles: dentro de un procesador, en varios procesadores dentro de la misma computadora, en varias computadoras dentro de un cluster, en varios clusters a través de Internet... Obviamente, estas técnicas son muy distintas entre sí, aunque comparten la misma filosofía.

Para que un código sea paralelizable, debe ser posible dividir el problema que resuelve en subproblemas más pequeños, de forma que puedan ser ejecutados en distintos elementos de proceso. Los códigos de Monte Carlo son idóneos para ser paralelizados, al efectuar operaciones independientes sobre un conjunto de datos de entrada.

En la computación distribuida, un programa es dividido en fragmentos que se ejecutan en múltiples computadoras comunicadas a través de una red. En general, estos programas tienen que tratar con entornos heterogéneos, redes de diferentes latencias, y fallos impredecibles en la red o las computadoras.

La computación distribuida se divide en dos grandes grupos: Alta Productividad (*High-Throughput Computing, HTC*) y Alto Rendimiento (*High-Performance Computing, HPC*), que suponen aproximaciones radicalmente diferentes y son aplicables a distintos entornos.

Como regla general, los sistemas de alto rendimiento ejecutan aplicaciones fuertemente acopladas, esto es, con muchas dependencias de datos entre las tareas que se ejecutan en los distintos nodos. Por tanto, es conveniente ejecutarlas en un único *site* equipado con interconexiones de baja latencia. En el otro extremo están los sistemas de alta productividad, que ejecutan trabajos secuenciales independientes. Estos trabajos pueden ser planificados en muchos recursos computacionales, e incluso en diferentes dominios administrativos.

Las tareas de alto rendimiento se caracterizan por necesitar una gran cantidad de recursos computacionales (capacidad de cómputo o consumo de memoria) durante períodos relativamente cortos de tiempo. Por otro lado, las aplicaciones de alta productividad, al estar compuestas por pequeñas tareas independientes, no necesitan una gran cantidad de recursos en un momento y *site* dado, sino que su ejecución puede ser distribuida o fragmentada según las necesidades del momento.

3.1.1. Computacion en Cluster

Un cluster es un conjunto de ordenadores de sobremesa o estaciones de trabajo, unidos mediante una red de alta velocidad, de modo que el conjunto es visto como un sólo equipo. Habitualmente se conectan al exterior mediante un solo equipo, y disponen de gestores de carga para distribuir el trabajo entre todos sus componentes de una manera eficiente.

Los clusters, además de poder realizar tareas de alto rendimiento o alta productividad, son especialmente recomendables para equipos en los que se necesite una alta disponibilidad, es decir, garantizar un funcionamiento ininterrumpido del mismo. Estos clusters disponen de un software específico que detecta fallos y asegura su corrección, a la vez que su configuración hardware evita el tener un único punto que pudiera fallar.

3.1.2. Paso de Mensajes

El paso de mensajes es una técnica empleada en programación concurrente para aportar sincronización entre procesos, transmitir los datos y permitir la exclusión mutua. Su principal característica es que no necesita memoria compartida, por lo que es muy empleada en la comunicación para sistemas distribuidos o, en general, nodos independientes comunicados a través de algún medio.

Empleando paso de mensajes, cada proceso se ejecuta independientemente, y sólo se comunica con otros procesos cuando haya instrucciones que lo soliciten. Esta comunicación, que puede ser punto a punto o *broadcast*, se realiza a través de pequeños paquetes denominados mensajes que contienen la información a intercambiar. Un sistema de paso de mensajes, tales como MPI o SHMEM (detallados en los siguientes apartados), proporciona funciones básicas llamadas primitivas que el programador empleará para transmitir la información.

En general, el paso de mensajes funciona mejor cuanto más débilmente acoplada es la aplicación. Cada proceso se ejecutará, normalmente, en un sólo procesador y área de memoria, de modo que la comunicación entre procesos se realiza únicamente cuando sea necesario.

MPI

MPI (*Message Passing Interface*) es una especificación de una librería estándar para el paso de mensajes definida por el MPI Forum. La primera versión fue publicada en mayo de 1994 [11] y la segunda en 1996 [13].

Con MPI el número de procesos requeridos se asigna antes de la ejecución del programa, y no se crean procesos adicionales mientras la aplicación se ejecuta. A cada proceso se le asigna un identificador, que permite determinar qué fracción del código ha de ejecutarse.

MPI incluye comunicaciones punto a punto y colectivas (globales), todas orientadas a un grupo de procesos especificado por el usuario.

Las llamadas de MPI se dividen básicamente en 2 clases:

- Envío y recepción de datos. Estas llamadas permiten a un proceso recuperar la información calculada por otro u otros procesos. En el caso de las comunicaciones punto a punto, se obtiene la variable deseada del proceso deseado, identificado por su *rank*. En

el caso de las comunicaciones colectivas, se pueden obtener datos simultáneamente de todos los procesos.

La figura 3.1 muestra dos de estas funciones. MPI_GATHER recoge de todos los procesos el valor de una determinada variable y la almacena en un array dentro del proceso que la invocó. MPI_REDUCE recoge el valor de una determinada variable y efectúa una operación matemática, en este caso una suma. De nuevo, el valor se almacena en una variable en el proceso que la invocó.

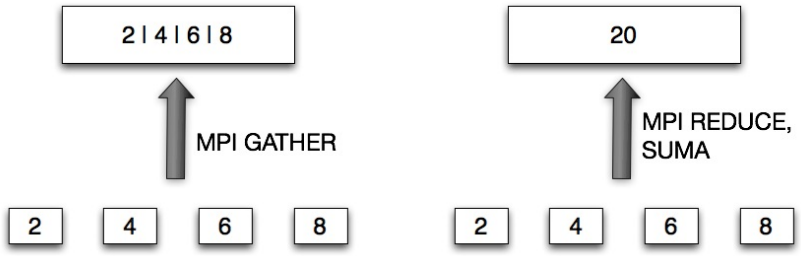


Figura 3.1: Funciones MPI_GATHER y MPI_REDUCE

- Control de las comunicaciones. Aquí se incluyen todos los mecanismos necesarios para sincronizar los procesos y permitir la exclusión mutua. La figura 3.2 representa gráficamente la función MPI_BARRIER, que implementa una barrera.

SHMEM

SHMEM es un una librería propiedad de Silicon Graphics, Inc. [1] que engloba un conjunto de funciones y subrutinas para extender las posiblidades de MPI-1, las cuales funcionan en los computadores con IRIX y Linux. Este mecanismo puede ser implementado de manera muy eficiente en equipos con memoria compartida (*UMA* o *NUMA*).

Esta librería incluye, además de las operaciones de paso de mensajes, una serie de llamadas que aumentan su velocidad de ejecuión. Cabe destacar un mecanismo de punteros globales para acceder a datos de otro proceso mediante operaciones *load/store*, y una serie de rutinas muy optimizadas para operaciones multipunto tales como reducciones globales.

La razón de la alta velocidad de ejeución de SHME es que está muy cerca del hardware. Esto requiere una mayor atención del programador en áreas como la sincronización, pero el rendimiento obtenido es mayor.

SHMEM hace una copia memoria-memoria directa, que es el mecanismo más rápido para mover datos en un computador CRAY T3E. Esta es la arquitectura en la que hasta ahora corría FAFNER-2, por lo que utilizar este mecanismo de paso de mensajes es bastante razonable.

El principal inconveniente es que SHMEM no es portable. Por lo tanto, ha sido desplazado por MPI, siendo recluido a software antiguo en máquinas no actualizadas. En concreto, SHMEM no funciona en máquinas X86 bajo Linux, lo que imposibilita su uso en redes Grid (compuestas casi exclusivamente por este tipo de equipos).

3.2. Computación Grid

3.2.1. Qué es la Grid

El problema que la Grid intenta resolver es *coordinar la compartición de recursos y resolución de problemas en organizaciones virtuales dinámicas y multi-institucionales* [16]. Por supuesto, esta compartición tiene que estar altamente controlada, de modo que tanto los usuarios como los proveedores de servicios sepan claramente qué, cuánto, a

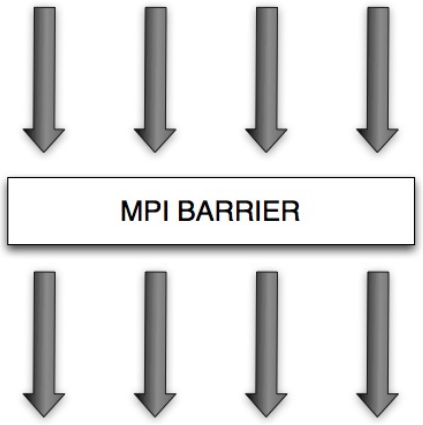


Figura 3.2: Función MPI_BARRIER

quién y bajo qué condiciones se comparte un determinado recurso. Estas condiciones definen grupos de usuarios denominados *organizaciones virtuales* (VOs).

Con esto en mente, una definición intuitiva de la Grid sería: mientras que la World Wide Web es un servicio para compartir información en Internet, la Grid es un servicio para compartir recursos de computación y capacidad de almacenamiento en Internet.

Una definición más clásica se puede encontrar en [12]. Según esto, una infraestructura Grid sería un sistema que:

- Coordina recursos que no están sujetos a un control centralizado.
- Usa protocolos e interfaces estándar, abiertos y de uso general.
- Proporciona calidades de servicio no triviales.

El primer punto implica que los recursos no pertenecen a una misma persona u organización, por lo que no se tiene un control absoluto de los mismos. Por el contrario, el uso de dichos recursos está supeditado a la autorización del administrador de los mismos, así como el software instalado, política de seguridad, o cualquier otro criterio.

El segundo punto deja claro que la Grid pretende ser un entorno de uso general, no orientado a un determinado tipo de aplicación o ámbito -tal como la compartición de ficheros, por ejemplo-. Para ello es fundamental que sea un entorno abierto, de modo que cada cual pueda adaptarlo a sus necesidades, y pueda ser implementado en el mayor número de arquitecturas y sistemas posible. El hecho de que los protocolos sean estándar evita la fragmentación, es decir, que haya diferentes redes Grid utilizando distintos protocolos y no puedan colaborar.

Y el tercer punto implica que la Grid *funcione*. De nada sirve todo lo demás si no somos capaces de proveer una infraestructura estable, en la que sus usuarios puedan confiar, y que proporcione una utilidad mayor que la de la suma de sus partes. Después de todo una Grid no es un fin en sí mismo, sino un medio que proporcione a sus usuarios los recursos informáticos que estos necesiten para llevar a cabo sus tareas.

3.2.2. Globus Toolkit

Globus Toolkit[2] es el estándar de facto en la computación grid. Está compuesto por un conjunto de servicios, librerías y herramientas de desarrollo diseñadas para construir aplicaciones basadas en la

Grid [9]. El diseño de la versión 4, GT4, se muestra en la figura 3.3, y está detallado en los siguientes apartados.

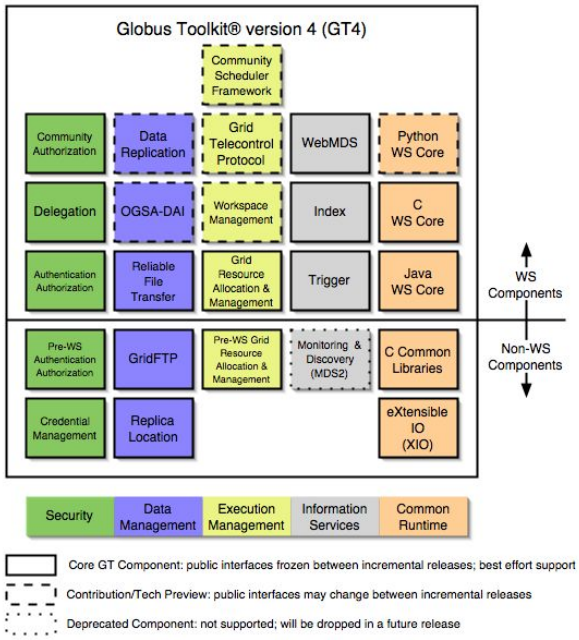


Figura 3.3: Arquitectura de Globus Toolkit 4

Seguridad

Los componentes de seguridad de Globus Toolkit 4, *Grid Security Infrastructure* (GSI) se encargan de proporcionar comunicaciones seguras y una política de seguridad común en los distintos *sites*. En concreto, sus responsabilidades son:

- *Autenticación y Autorización*: controlar quién puede acceder a cada recurso o servicio Grid; gestionar la existencia de VOs, así como la política de seguridad de cada una de ellas.
- *Manejo y delegación de credenciales*: crear credenciales de seguridad de los usuarios que no tengan acceso a una autoridad de certificación (CA) externa; delegar las credenciales de seguridad del usuario de forma temporal.

Manejo de datos

Proporciona las herramientas necesarias para el manejo de datos dentro de una infraestructura Grid.

- *Grid File Transfer Protocol (GridFTP) y Reliable File Transfer Service (RFT)*: GridFTP es un servicio FTP seguro; RFT es un servicio web de transferencia de ficheros que utiliza GridFTP para conseguir el mejor rendimiento posible, teniendo en cuenta las necesidades particulares de la Grid (grandes cantidades de datos en entornos dinámicos).
- *Réplica de datos*: asegurarse de que los datos están replicados en diferentes servidores para evitar su pérdida, y permitir al usuario la localización de los mismos de manera eficiente.

Control de la ejecución

Estos componentes de GT4 se encargan del lanzamiento, planificación y monitorización de los trabajos. Los más importantes son:

- *Grid Resource Allocation and Management (GRAM)*: proporciona servicios para lanzar y monitorizar trabajos en la Grid.
- *Community Scheduler Framework (CSF)*: proporciona una interfaz única para acceder a los gestores de carga locales, tales como *Portable Batch System (PBS)* o *Sun Grid Engine (SGE)*.

Servicios de Información

Los servicios de información de GT4 son un conjunto de herramientas para monitorizar y descubrir recursos dentro de una organización virtual. Están compuestos por un *Index Service*, utilizado para recuperar información de los distintos recursos de una VO, y un *Trigger Service*, que además de recuperar los información de los recursos, puede desempeñar determinadas tareas basándose en esos datos.

3.2.3. GridWay

Como se ha detallado en el apartado anterior, Globus Toolkit proporciona un conjunto de herramientas que permiten al usuario utilizar una infraestructura grid. Sin embargo, este usuario es responsable de efectuar manualmente todos los pasos necesarios para obtener una funcionalidad: decidir dónde se ejecutará el trabajo, enviarlo, recoger los datos de salida... Más aun, Globus Toolkit no proporciona soporte para la migración de trabajos ni la recuperación de fallos, un aspecto clave a la hora de trabajar eficientemente.

El hecho de que las redes Grid sean entornos enormemente dinámicos hace que las aplicaciones que dependen de ellas deban ser capaces de adaptarse a un entorno cambiante, para poder aprovechar los nuevos recursos que aparecen y recuperarse de posibles fallos en aquellos en los que se esté ejecutando. Para ello, en la literatura [15] se han propuesto dos técnicas fundamentales:

- *Planificación adaptable* que envíe trabajos a los recursos Grid teniendo en cuenta sus recursos físicos, disponibilidad, y funcionamiento durante los trabajos previamente enviados a ese recurso.
- *Ejecución adaptable* que pueda migrar trabajos entre diferentes recursos teniendo en cuenta tanto eventos internos de la aplicación (necesidades cambiantes) como de la Grid (aparición de nuevos recursos o desaparición de los existentes).

GridWay es una meta-planificador que permite una ejecución sencilla de trabajos en un entorno Grid dinámico, llevando a cabo todos los pasos necesarios [14]. El objetivo final es que el usuario final pueda definir sus necesidades de una manera sencilla, delegando a GridWay la responsabilidad de que el trabajo sea ejecutado de una manera eficientemente, empleando todos los recursos posibles. De este modo la utilización de una infraestructura Grid deja de estar restringida a usuarios expertos, y un abanico mucho más amplio de personas puede utilizarla para satisfacer sus necesidades de computación.

Para lograr este objetivo, el núcleo de GridWay es un agente que realiza todas las labores de planificación y supervisa que la ejecución del trabajo sea correcta y eficiente. La adaptación a condiciones cambiantes se realiza mediante una planificación y ejecución adaptables. En concreto, una vez que un trabajo ha sido enviado a un recurso en concreto puede ser migrado por las siguientes razones:

- Relacionadas con la Grid: un recurso mejor es descubierto, el recurso original falla, o el trabajo es cancelado o suspendido por el administrador del recurso.
- Relacionadas con la aplicación: se produce una disminución del rendimiento, cayendo por debajo de unos límites establecidos, o los requisitos de la aplicación cambian.

En este proyecto el objetivo final es que la aplicación gridificada pueda ser empleada por usuarios sin conocimientos técnicos de Grid. En este contexto, GridWay supone una importante herramienta, ya que permite que el usuario se centre en el problema a resolver y no en la tecnología subyacente.

3.2.4. EGEE

El proyecto *Enabling Grids for E-sciencE*, EGEE [3] reúne a científicos e ingenieros de más de 240 instituciones en 45 países alrededor del mundo para construir una infraestructura Grid con fines científicos que esté disponible permanentemente. Después de cuatro años de existencia, en el 2008 entró en su tercera fase de desarrollo.

Sus objetivos se centran en tres áreas clave:

- Construir una infraestructura Grid consistente, robusta y segura.
- Mantener y mejorar el *middleware*, de forma que se proporcione a los usuarios la mejor experiencia de uso posible.
- Atraer nuevos usuarios tanto de la industria como del entorno científico, y asegurarse de que reciban toda la educación y soporte técnico necesarios que necesiten para emplear los recursos eficientemente.

Actualmente, EGEE cuenta con una impresionante cantidad de recursos: más de 50.00 CPUs, más de 15 Petabytes de almacenamiento y una tasa de transferencia de más de 1.5 GB/s, que permite la ejecución de 150.000 trabajos al día. Para ello dispone de un capital humano de más de 9000 personas/mes, y un presupuesto de 52 millones de Euros en esta tercera fase(de los que 37 son subvenciones de la Comisión Europea).

En EGEE, hay aproximadamente 130 organizaciones virtuales con diferentes miembros, reglas a la hora de enviar trabajos y recursos accesibles. De todas ellas, se utilizaron los recursos de dos a la hora de elaborar este trabajo: *dteam* durante el desarrollo del software (es una cola donde se permiten las pruebas y trabajos que no sean de producción) y *fusion* una vez que la aplicación estaba en producción, con muchos más recursos que *dteam* y una infraestructura más estable.

En el momento de redactar esta memoria, la VO *fusion* disponía de 45 nodos con más de 15.000 CPUs disponibles, de las que un tercio pertenecían al GridPP, *U.K. Computing for Particle Physics*. En este contexto, FAFNER-2 pasará a formar parte de esta VO junto a otras aplicaciones centradas en el cálculo masivo de trayectorias, el transporte cinético o la optimización de dispositivos experimentales de fusión.

Capítulo 4

Implementación

El proceso de gridificación de FAFNER-2 se llevó a cabo en tres pasos.

El primero consistió en la transformación del mecanismo de paso de mensajes, de SHMEM a MPI, dentro de una máquina SGI de memoria compartida. Después, portamos el código a un cluster X86. Y por último, ejecutamos este código en la Grid. El resto de este capítulo está dedicado a detallar este proceso.

4.1. Actualización de la Paralelización: de SHMEM a MPI

El primer paso de este trabajo consistió en una transformación en el mecanismo de paso de mensajes, de SHMEM a MPI. Esto es necesario ya que el programa original sólo se podía ejecutar en máquinas con una arquitectura muy específica (equipos de memoria compartida corriendo IRIX o versiones concretas de Linux), lo que limitaba enormemente sus posibilidades de ejecución en Grid.

4.1.1. Entrada/Salida

La entrada/salida funciona de manera ligeramente distinta en SHMEM y MPI. A la hora de mostrar información por pantalla y/o escribir en ficheros, fue necesario modificar el código para que el resultado fuera exactamente el mismo. Esto es necesario porque los ficheros generados por FAFNER-2 son la entrada de otras aplicaciones, y cualquier modificación haría que dejaran de funcionar correctamente.

4.1.2. Equivalencia entre Funciones

Como se explicó en la sección 3.1.2, la librería SHMEM está basada en MPI, y en muchos casos sus funciones son equivalentes. La tabla 4.1 detalla estas equivalencias, en donde se han simplificado los prototipos de las funciones para facilitar su legibilidad.

SHMEM	MPI	Función
START_PES	MPI_INIT	Inicia el paralelismo
NUM_PES	MPI_COMM_SIZE	Número de hilos
MY_PE	MPI_COMM_RANK	Identifica un hilo
SHMEM_BARRIER_ALL	MPI_BARRIER	Barrera (fig. 3.2)
SHMEM_COLLECT	MPI_GATHER	Ver fig. 3.1

Cuadro 4.1: Equivalencia entre las funciones de SHMEM y MPI

4.1.3. Optimización del Código

En otras partes de la aplicación, el código fue optimizado evitando llamadas innecesarias a funciones MPI. Para ello se analizó el código y fue posible optimizar el flujo de información, utilizando las funciones MPI más adecuadas. Entre otras, se utilizó en varias ocasiones la función MPI_REDUCE(fig. 3.2) en sustitución de SHMEM_COLLECT seguido de un bucle para operar con el array resultado.

Como consecuencia de esta optimización, el tiempo de ejecución se redujo ligeramente. Los resultados obtenidos están detallados en la sección 5.1.

4.2. Actualización de la Arquitectura: de SGI a X86

Tras portar el código a MPI, el siguiente paso consistió en portar la aplicación a X86. Y aunque desde a primera vista esta tarea pueda parecer sencilla (y, de hecho, debería serlo) dista de ser trivial.

Para empezar, existen enormes diferencias entre las distintas versiones de Fortran y MPI. Esto provoca que el código que tiene un correcto

funcionamiento con un determinado compilador en un determinado entorno de ejecución, al cambiar de compilador, plataforma o versión de MPI se comporte de una manera completamente diferente.

Por otro lado, un pequeño número de rutinas empleadas en FAFNER-2 no pertenecen al standard de Fortran, sino a librerías implementadas para una determinada arquitectura y sistema operativo. Al migrar la aplicación en ocasiones dejan de estar disponibles, con los problemas que esto provoca. Pero otras veces, aun teniendo el mismo prototipo su funcionalidad es ligeramente distinta, con lo que la aplicación -que hasta ese momento estaba funcionando correctamente- comienza a tener errores aparentemente inexplicables.

Además, cada versión del compilador de Fortran y de las librerías de MPI tienen sus particularidades, dependiendo tanto de la versión como del fabricante. Por tanto es necesario disponer de varias versiones de ambos, y combinarlas hasta que encontrar la forma de que una aplicación concreta -que en otro equipo se podía ejecutar sin problemas- funcione.

Durante la realización de este trabajo, fue necesario el localizar todos estos conflictos e incompatibilidades y corregirlos, de forma que la aplicación tenga el comportamiento esperado. A continuación detallaremos los problemas a los que nos enfrentamos, así como su solución.

4.2.1. Librerías de SGI

FAFNER-2 originalmente utilizaba rutinas propietarias de SGI para Fortran. Al migrar el código a X86, fue necesario implementar estas rutinas de forma manual. Para ello, tras consultar el manual de referencia de estas rutinas [4], creamos un archivo con funciones que tienen el mismo prototipo, y funcionan de la misma manera.

La excepción son unas rutinas empleadas en las primeras versiones del programa, y que permitían mostrar la salida de FAFNER-2 de forma gráfica. Actualmente los usuarios del programa han dejado de utilizar esta funcionalidad de FAFNER-2, empleando programas de terceros más potentes y versátiles. Hemos decidido dejar de lado esta utilidad, y centrarnos en optimizar al máximo el rendimiento de la aplicación.

4.2.2. Librería NAG

El CIEMAT posee una licencia de uso de la librería NAG[5], una serie de funciones matemáticas altamente optimizadas que consiguen un alto rendimiento al procesar grandes cantidades de datos. Esta librería sólo está compilada para SGI, por lo que fue necesario recompilarla para X86 a partir del código fuente. Para reducir al máximo su tamaño únicamente se incluyeron aquellas funciones necesarias. De ese modo, pasamos de tener más de 180 rutinas a únicamente 5, reduciendo su tamaño en un 99 %.

4.2.3. Variables de Entorno

El tratamiento de las variables de entorno es radicalmente disinto en el equipo SGI de memoria compartida y en el clúster. En la máquina SGI, un programa Fortran MPI podía acceder a estas variables desde cualquiera de sus hilos. Bajo X86, sólo el primer hilo es capaz, por lo que una aplicación que haga uso de estas variables tiene que ser modificada.

Para ello, nuestra aproximación consistió en hacer que, al principio del programa, el primer hilo guardara el valor todas las variables de entorno que se fueran a necesitar en el código en ficheros temporales. Después, en el resto de la aplicación se reemplazaron las rutinas que leen variables de entorno por fragmentos de código que leen los ficheros temporales antes creados.

Dependiendo del tipo de aplicación y el lugar que ocupen en el código las consultas a variables de entorno, también es posible sustituirlas por un *broadcast* desde el primer hilo hacia todos los demás. En este caso se reducen los accesos a fichero, a cambio de comprometer la concurrencia. Elegir una opción u otra queda en manos del programador, que deberá evaluar la mejor opción según las características de su programa.

4.2.4. De 64 a 32 bits

En el problema físico a resolver (calcular la trayectoria de partículas neutras) la precisión de cálculo es de vital importancia. Por otro lado, dado que el programa realiza múltiples iteraciones sobre cada partícula

(para calcular su posición en diferentes momentos) cualquier pequeño error es automáticamente amplificado hasta unos valores inaceptables.

Originalmente, FAFNER-2 estaba pensado para ejecutarse en máquinas de 64 bits. Sin embargo, al ser portado a Grid es necesario que pueda ejecutarse en máquinas de 32 bits. Esto ocasiona problemas a varios niveles:

- Es necesario mantener un tamaño de datos, tanto para números reales como enteros, de 64 bits, por lo que hay que obligar al compilador a que ambos tipos de datos sean de 64 bits por defecto. Esto hace que determinadas variables no estén alineadas en memoria. Por ello, hay que modificar el código para evitarlo en la medida de lo posible y, además, hacer que el compilador fuerze el alineamiento.
- El hecho de que se almacenen variables de 64 bits en un computador de 32 bits reduce el rendimiento de la aplicación, dado que el procesador no está optimizado para este tipo de datos. Sin embargo, la posibilidad de utilizar procesadores más recientes y de una potencia mucho mayor compensa con creces este inconveniente.
- Es posible que el uso de punteros y *arrays* en los que el desplazamiento haya sido introducido de forma manual haga que la aplicación no funcione correctamente al cambiar el tamaño de datos de 32 a 64 bits. Por suerte FAFNER-2 no hace uso de esta técnica de programación, por lo que no se vio afectado por esta situación.

4.3. Gridificación

Por último, la aplicación fue gridificada. Los siguientes pasos fueron necesarios:

4.3.1. Subdivisión del Problema

Dado que la finalidad del Grid es poder lanzar múltiples ejecuciones simultáneas de la aplicación, es necesario subdividir el problema a resolver para que cada ejecución pueda encargarse de una sección.

Ya que FAFNER-2 es un código de Monte Carlo, no fue necesario modificar el algoritmo para subdividir el problema. Por el contrario, el

trabajo de gridificación se centró en subdividir los datos de entrada, para que cada ejecución resolviera una parte. Así, en nuestra aproximación, implementamos dos *wrapper*, uno que preprocesa los datos de entrada de la aplicación y otro que hace un post-proceso de los datos de salida.

En primer lugar, en el *wrapper* de preproceso creamos un algoritmo que genera semillas de números aleatorios del mismo modo que hace el programa. De ese modo cada ejecución de la aplicación era inicializada con una semilla distinta, lo que nos aseguraría que no estábamos repitiendo los mismos cálculos una y otra vez. Al utilizar el mismo algoritmo de generación que la aplicación original, se consigue que el resultado obtenido sea el mismo.

Tras ejecutar el programa en Grid y recoger los resultados, el *wrapper* de post-proceso se encarga de juntar todos los resultados parciales (ver 4.3.2). Para ello, efectúa los cálculos necesarios para unir los resultados de todas las ejecuciones. Por último genera los ficheros de salida, de manera que el resultado obtenido es el mismo que si el programa se hubiera ejecutado en un cluster local.

4.3.2. Resultados Parciales

La aplicación FAFNER-2 produce datos de salida en dos lugares distintos: a lo largo de la ejecución del programa mediante el hilo número cero, y tras juntar todos los hilos. Además, esta información ya está procesada cuando es mostrada en pantalla o escrita en un fichero de salida (por ejemplo, en el caso de una media,ésta ya está calculada). Eso no siempre servía en la gridificación, por lo que modificamos la aplicación para que ofreciera estos resultados antes de procesarlos.

Para ello, introdujimos en el código sentencias que vuelcan en un fichero de datos las variables y cálculos parciales deseados, en forma de XML. Así, además de tener la aplicación funcionando correctamente, producimos resultados que son utilizados para la gridificación (ver gráfico 4.1) . Estos ficheros, junto con la salida del programa, constituyen la entrada del *wrapper* de post-proceso. Este *wrapper* lee los ficheros XML y, junto con la información que necesita de los ficheros de salida (que también están en formato de texto plano), genera un nuevo fichero de salida. Este último fichero es equivalente al que se obtendría habiendo ejecutado el conjunto de datos de entrada en un sólo *site*, sin haberlo subdividido y enviado al Grid.

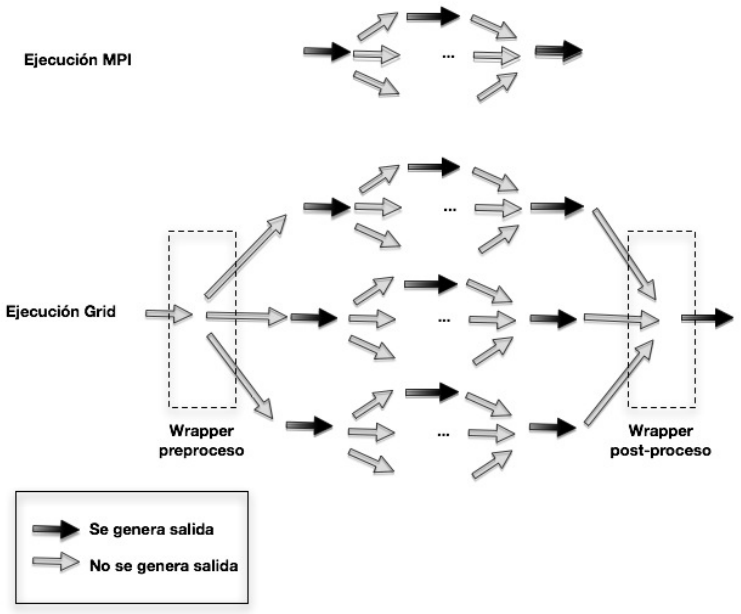


Figura 4.1: Generación de los ficheros de salida en las versiones MPI y Grid de FAFNER-2

Capítulo 5

Análisis del Rendimiento

El análisis del rendimiento de las distintas versiones de FAFNER-2 que se han obtenido en el desarrollo de este trabajo será llevado a cabo en tres pasos.

- Primero se estudiará la diferencia de rendimiento tras el cambio en el mecanismo de paso de mensajes, de SHMEM a MPI. Para ello ejecutaremos repetidamente ambas versiones de FAFNER-2 en la misma máquina, bajo diferentes condiciones de carga, con el fin de estudiar su comportamiento.
- A continuación, tras migrar la aplicación de la arquitectura MIPS a X86, se analizará el rendimiento en esta última, para compararlo con los resultados obtenidos en la sección anterior. Este paso es muy importante, ya que FAFNER-2 se ejecutará en un cluster local cuando los fenómenos a simular sean de pequeña escala. Comprobaremos cómo el paso a un cluster de mayor número de nodos, y con procesadores más modernos, disminuye enormemente el tiempo de ejecución.
- Por último, tras gridificar la aplicación, mediremos el tiempo que tarda en ejecutarse en la Grid. Esto nos permitirá comprobar, además de la diferencia de tiempos de ejecución, la escalabilidad del programa. La ejecución en Grid no sólo pretende permitir que el mismo problema sea resuelto en menos tiempo, sino el que sea posible simular problemas más complicados que en un sólo cluster resultaban inabarcables.

En los siguientes apartados se detallarán los resultados obtenidos en este análisis.

5.1. SHMEM a MPI

5.1.1. Testbed

La primera parte de este trabajo, el paso de SHMEM a MPI, se realizó en el computador *Jen50*, un SGI Origin 3800.

Este es un equipo de memoria compartida, con 122 procesadores MIPS R14000 funcionando a 600 MHz. Tiene 126 GB de memoria compartida, y una gran cantidad de almacenamiento tanto en discos como cintas.

5.1.2. Resultados Obtenidos

La tabla 5.2 muestra el tiempo medio de ejecución de FAFNER-2 en sus versiones SHMEM y MPI dentro del cluster *Jen50*.

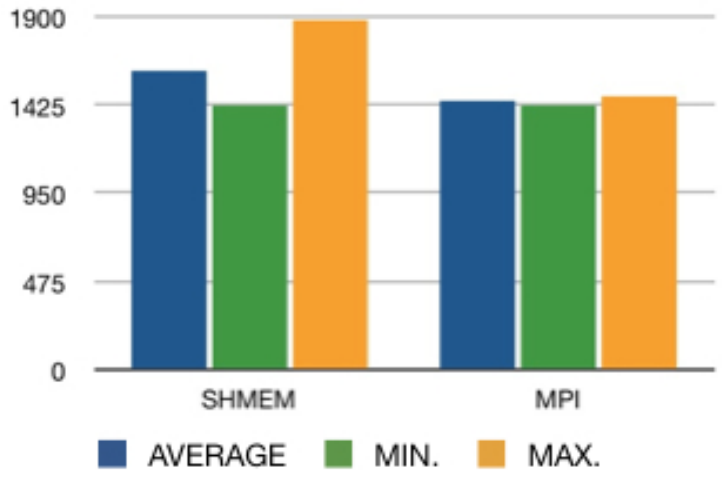


Figura 5.1: Tiempo de ejecución en *Jen50*, versiones SHMEM y MPI

Como se puede observar, al migrar la aplicación de SHMEM a MPI y optimizarla minimizando su tráfico de red, hemos reducido el tiempo medio de ejecución en 149 segundos, algo más de un 10 %. Además, como un efecto positivo con el que no contábamos, se ha reducido la variabilidad en los tiempos de ejecución: la diferencia entre el mayor y el menor tiempo ha pasado de 458 a 51 segundos.

5.2. SGI a X86

5.2.1. Testbed

Para evaluar el aumento de rendimiento en el cambio de arquitectura utilizamos dos máquinas. La primera es *Jen50*, utilizando los datos obtenidos en el apartado anterior. La segunda es un clúster heterogéneo llamado *Lince*, descrito en la tabla 5.1.

Tipo de nodos	Características
Infiniband, 32 nodos (64 slots)	Dual Xeon 3.2 GHz, 2 GB Hyperthreading Desactivado Soporte Infiniband
Serial, 46 nodos (184 slots)	Dual Xeon 3.2 GHz,2 GB 1Gb/s Ethernet
Serial2, 10 nodos (40 slots)	Dual Xeon dual-core 3.0 GHz 2 GB 2Gb/s Ethernet (Bonding)

Cuadro 5.1: Resumen de las características del cluster *Lince*

FAFNER-2 se ejecutó en los nodos *Serial*. El motivo es que los nodos con Infiniband están reservados para trabajos HPC con mucha comunicación entre hilos, para aprovechar al máximo la capacidad de la red de comunicaciones. FAFNER-2, como se ha comentado, tiene poca comunicación entre los hilos, por lo que no es necesario disponer de esta red.

Para este análisis del rendimiento, lanzamos la aplicación con 32 hilos de ejecución, los mismos que en el apartado anterior. De este modo, sabemos que el aumento de rendimiento es debido al cambio de plataforma y no a un mayor número de nodos. También comprobamos el máximo número de nodos en el que una aplicación MPI funciona correctamente en este equipo, determinando así su escalabilidad.

5.2.2. Resultados Obtenidos

El resultado obtenido al ejecutar FAFNER-2 sobre 32 *slots* de *Lince* se muestra en la figura 5.2.

Como se puede ver, el tiempo de ejecución de la aplicación apenas ha disminuido. Esto es un resultado inesperado, ya que se trata de una

máquina mucho más reciente. Con los datos teóricos de rendimiento, cabía esperar una importante mejora, no de un escaso 3 %.

Los motivos de que la mejora sea tan pequeña apuntan al uso de hyperthreading (enviando varios procesos al mismo procesador) en los procesadores del cluster *Lince*, y que éstos sean de 32 bits. En cualquier caso, un análisis más profundo de este hecho está fuera del alcance de este trabajo, y se ha propuesto como trabajo futuro. Como ya se ha comentado, es importante que la versión MPI de FAFNER-2 funcione de una manera eficiente en este cluster, ya que será empleada para la realización de experimentos pequeños. Por tanto, este problema tendrá que ser analizado en profundidad y solucionado.

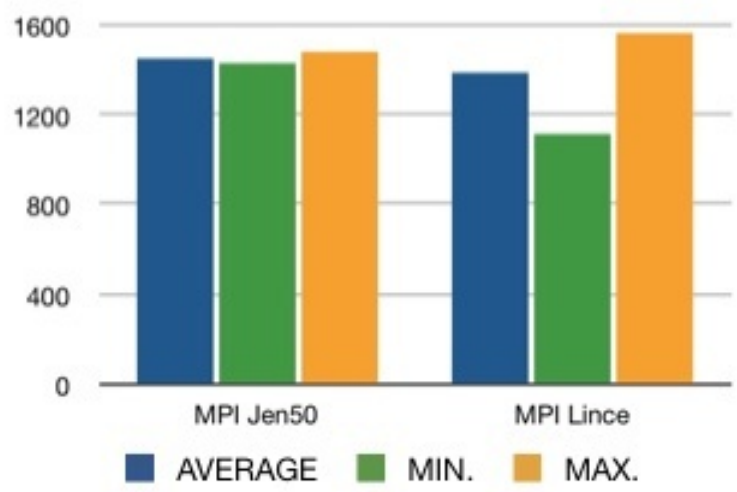


Figura 5.2: Tiempo de ejecución de FAFNER-2 en *Jen50* y *Lince* con 32 hilos de ejecución

Como nota positiva, cabe decir que en este cluster la aplicación se ejecutó correctamente en 48 *slots* simultáneamente, lo que supone una mejora de cerca de un 50 % en su tiempo de ejecución.

5.3. Cluster a Grid

Medir el rendimiento de la aplicación en Grid resulta complicado. Dependiendo de los parámetros elegidos, equipos a los que se envía el trabajo, y momento elegido para la ejecución, los resultados son tremendamente variables.

Además, hay que tener en cuenta que la infraestructura del EGEE está orientada a aplicaciones en producción, por lo que no se puede utilizar en la etapa de desarrollo. Por eso, en esta fase utilizamos únicamente un nodo grid del CIEMAT. De este modo, en el caso de ocurrir cualquier imprevisto es posible detener la ejecución desde dentro del nodo sin ocasionar mayores problemas. Por otro lado, al conocer la carga de sistema, podemos enviar el trabajo en los momentos que la máquina esté siendo poco utilizada, y así no afectar al resto de los trabajos.

Todos los nodos Grid del EGEE utilizan un mismo sistema operativo, el Scientific Linux CERN [6], y han de tener determinadas librerías y software disponibles. Esto asegura que las aplicaciones que funcionan correctamente en un nodo podrán funcionar en todos los demás. Por eso, el desplegar FAFNER-2 en el nodo Grid del CIEMAT nos asegura que, cuando la aplicación pase al estado de producción, funcione correctamente en toda la infraestructura.

5.3.1. Testbed

Para medir el rendimiento de FAFNER-2 en Grid, empleamos dos tipos de recurso.

El primero, para obtener el tiempo de ejecución, fue el nodo Grid del CIEMAT, llamado *ce-eela.ciemat.es*. Este equipo es un cluster con 20 nodos Intel Dual Xeon a 3.20 GHz, equipados con 2 GB de memoria.

Por otro lado, utilizamos nodos Grid remotos para comprobar el tiempo de transferencia de ficheros. Para ello empleamos 3 nodos diferentes del EGEE, situados a distancia del CIEMAT: uno de la Facultad de Informática de la UCM, otro de la Universidad Politécnica de Valencia, y un tercero de la Universidade Federal do Rio de Janeiro, Brasil.

5.3.2. *Wrapper* de pre-proceso

El *wrapper* de preproceso es el encargado de generar algunos de los ficheros de entrada de FAFNER-2, de modo que cada ejecución que se envíe a la Grid ejecute una parte del problema a resolver.

Su funcionamiento es relativamente simple, y su ejecución muy rápida. El tiempo de ejecución es de menos de un segundo.

5.3.3. Transferencia de ficheros

Para medir el tiempo de transferencia de los ficheros de entrada, realizamos el envío de datos a distintos nodos grid. Como se puede ver en la tabla 5.2, los resultados varían enormemente según las características del nodo elegido, así como su carga. Para minimizar la variación producida por la carga del nodo realizamos las medidas en todos los nodos a la vez, y a una hora del día donde la carga era mínima.

Estos ficheros incluyen el ejecutable de FAFNER-2, las librerías con la geometría del TJ-II y cierto número de archivos con los datos de entrada y especificación del problema a resolver. Ocupan un total aproximado de 50 MB.

El tamaño de los ficheros de salida de FAFNER-2 depende directamente del tamaño de la simulación realizada, así como de la precisión que se desee obtener. En este caso, estos ficheros ocupan un total de 17 MB.

El tiempo de transferencia de estos ficheros está también reflejado en la tabla 5.2.

Nodo grid	Ficheros de entrada	Ficheros de salida
aquila.dacya.ucm.es	9.16 s	3.28 s
ramses.dsic.upv.es	60.8	19.3 s
ce-biof.eela.ufrj.br	438.75 s	151 s

Cuadro 5.2: Tiempo de transferencia de los datos de entrada y salida de FAFNER-2 en Grid

Es importante destacar que los nodos en los que hemos ejecutado estas pruebas de tiempo no son en los que se ejecutó el programa. La razón, como hemos comentado antes, es el hecho de que FAFNER-2 no esté en producción. El envío de trabajos al EGEE está muy restringido, por lo que resulta complicado efectuar medidas de tiempo. Sin embargo, la transferencia de ficheros relativamente pequeños no supone un problema, y permite estimar el tiempo que se empleará cuando la aplicación vaya a ejecutarse en esta infraestructura.

5.3.4. Tiempo de ejecución

El tiempo de ejecución de FAFNER-2 lo medimos en el nodo grid del CIEMAT, llamado *ce-eela.ciemat.es*. De este modo podemos, a la vez de

enviar el trabajo, observar desde dentro del nodo la correcta ejecución del mismo. Los resultados obtenidos pueden verse en la figura 5.3

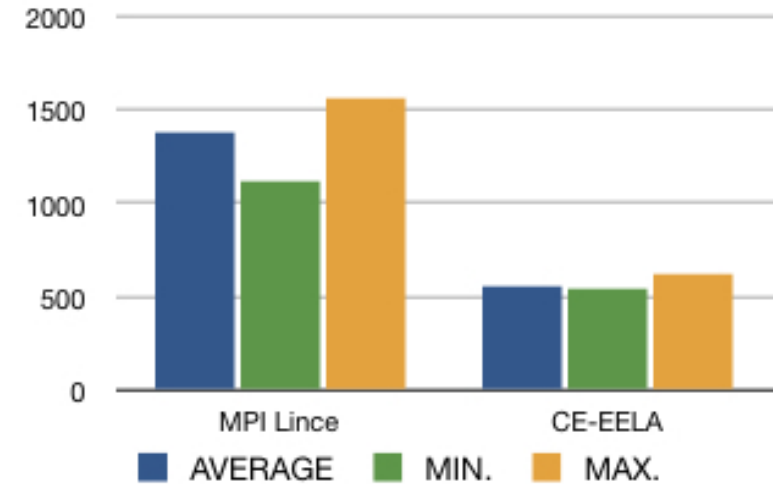


Figura 5.3: Tiempo de ejecución de FAFNER-2 en *Lince* y *CE-EELA*

El rendimiento obtenido en esta fase es muy positivo, consiguiendo una disminución del tiempo de ejecución de un 60 % dentro de un solo nodo grid. Estas medidas habrán de completarse cuando FAFNER-2 pase al estado de producción y pueda ser ejecutado en decenas de nodos del EGEE simultáneamente, así como cuando se trabaje con un número mayor de partículas.

5.3.5. *Wrapper* de post-proceso

El *wrapper* de post-proceso es el encargado de recoger los datos de salida de las distintas ejecuciones de FAFNER-2 en Grid, y operar con ellos para obtener el resultado final. Aunque no es una aplicación complicada, tiene que leer ficheros bastante voluminosos, extraer la información relevante y realizar distintas operaciones matemáticas. Su tiempo de ejecución depende del tamaño del problema ($n\frac{1}{4}$ de partículas). En este caso, tomó alrededor de 20 segundos en ejecutarse.

Capítulo 6

Trabajo Futuro

FAFNER-2 es una aplicación que continuará evolucionando los próximos años. Además de seguir siendo utilizada en el CIEMAT para simular los procesos que ocurren en el TJ-II -tanto sobre Grid como en un cluster con MPI-, se espera que sirva para simular el calentamiento por NBI en el ITER [7], el mayor proyecto de Fusión de la historia de la humanidad. Para ello, será necesario adaptar su código, proporcionando una interfaz clara y sencilla para poder elegir la geometría del reactor simulado mediante un fichero externo.

En este último entorno, la parte MPI del código será transformada, haciendo que FAFNER-2 se pueda ejecutar en cualquier *site* grid, no sólo en los que utilizan MPI. De este modo, se ampliará enormemente el número de máquinas del EGEE donde la aplicación puede correr.

Respecto a la versión MPI de FAFNER en el cluster *Lince*, se estudiará el por qué de la poca mejora de rendimiento obtenida. Si es posible se solucionarán los problemas que pudiera haber, con el fin de conseguir un mayor rendimiento.

Paralelamente a esta tarea, es necesario estudiar el flujo de información de la aplicación, de modo que se minimice la cantidad de datos que son transmitidos por Internet a los distintos nodos grid. De ese modo, se reduce el tiempo de ejecución, además de exponerse lo menos posible a la posibilidad de un fallo en la transmisión. Para llevar a cabo esta tarea es necesario implementar un *workflow*, estudiando *qué* datos es necesario enviar *dónde* y en *qué* momento.

Cuando estas tareas estén realizadas, se creará una interfaz web para el manejo de FAFNER-2 y análisis de sus resultados, de modo que se

simplifique al máximo el trabajo de los científicos que deseen usar la aplicación. Esta interfaz permitirá, además, el acoplar FAFNER-2 a otros códigos como ISDEP, creando *workflows* complejos que simulen un abanico de fenómenos del plasma.

Capítulo 7

Conclusiones

En la realización de este trabajo hemos logrado tres objetivos fundamentales: un cambio en la herramienta de paso de mensajes, de SHMEM a MPI; portar la aplicación a una arquitectura más moderna, de MIPS a X86; y por último, gridificar la aplicación.

La primera tarea, el cambiar la herramienta de paso de mensajes, a pesar de ser una tarea larga fue resuelta satisfactoriamente, consiguiendo que la aplicación funcione correctamente.

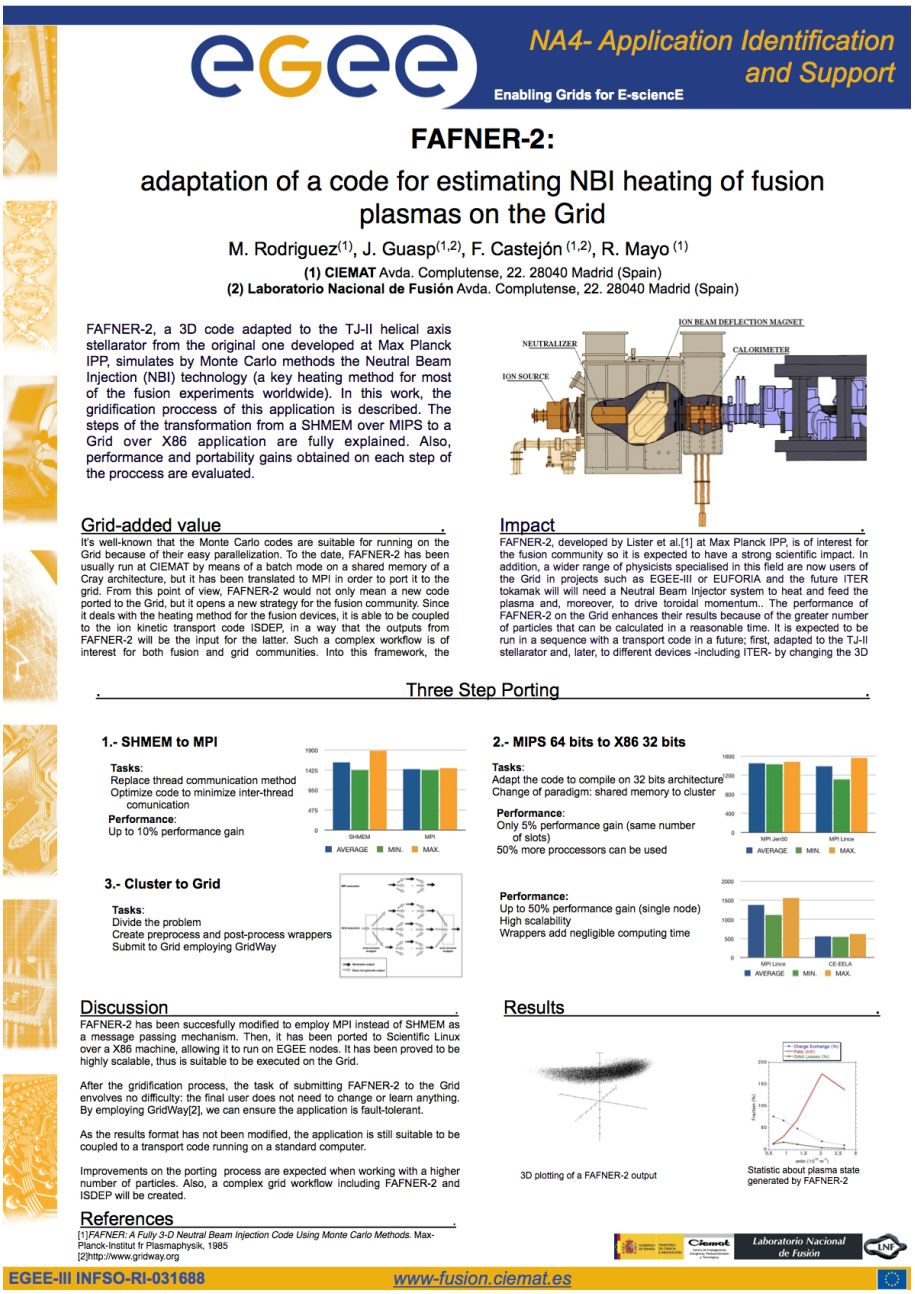
La segunda tarea, portar la aplicación de MIPS a X86, a pesar de parecer la más sencilla resultó la más complicada. A los problemas derivados del cambio de sistema operativo y software disponible se unieron los provocados por la utilización de un procesador de 32 bits. Todo esto provocó que fueran necesarios cambios profundos en la aplicación, desde la entrada/salida al generador de números aleatorios (por citar tan sólo algunos). Además, el hecho de que la aplicación no pueda correrse en modo interactivo sino que haya que enviarla a un sistema de colas *batch* dificulta enormemente la depuración, alargando el proceso y haciéndolo enormemente tedioso. Sin embargo, este paso es necesario para poder gridificar la aplicación, con lo que no es posible evitarlo.

Por último, se comprobó que FAFNER-2 funciona correctamente en Grid. Desarrollamos *wrappers* de pre-proceso y post-proceso, que generan los archivos de entrada necesarios y generan los de salida. Realizamos las medidas de tiempo necesarias, y comprobamos que la aplicación es escalable.

Apéndice A

Ponencias que ha originado este trabajo

Este trabajo se ha presentado en la conferencia "EGEE 08", celebrada en Estambul entre los días 22 y 26 de septiembre del 2008, con el título "FAFNER-2: adaptation of a code for estimating NBI heating of fusion plasmas on the Grid".



NA4- Application Identification and Support

Enabling Grids for E-sciencE

FAFNER-2:

adaptation of a code for estimating NBI heating of fusion plasmas on the Grid

M. Rodriguez[1], J. Guasp[1,2], F. Castejón [1,2], R. Mayo [1]

(1) CIEMAT Avda. Complutense, 22. 28040 Madrid (Spain)

(2) Laboratorio Nacional de Fusión Avda. Complutense, 22. 28040 Madrid (Spain)

FAFNER-2, a 3D code adapted to the TJ-II helical axis stellarator from the original one developed at Max Planck IPP, simulates by Monte Carlo methods the Neutral Beam Injection (NBI) technology (a key heating method for most of the fusion experiments worldwide). In this work, the gridification proccess of this application is described. The steps of the transformation from a SHMEM over MIPS to a Grid over X86 application are fully explained. Also, performance and portability gains obtained on each step of the proccess are evaluated.

Grid-added value

It's well-known that the Monte Carlo codes are suitable for running on the Grid because of their easy parallelization. To the date, FAFNER-2 has been usually run at CIEMAT by means of a batch mode on a shared memory of a Cray architecture, but it has been translated to MPI in order to port it to the grid. From this point of view, FAFNER-2 would not only mean a new code ported to the Grid, but it opens a new strategy for the fusion community. Since it deals with the heating method for the fusion devices, it is able to be coupled to the ion kinetic transport code ISDEP, in a way that the outputs from FAFNER-2 will be the input for the latter. Such a complex workflow is of interest for both fusion and grid communities. Into this framework, the

Impact

FAFNER-2, developed by Lister et al.[1] at Max Planck IPP, is of interest for the fusion community so it is expected to have a strong scientific impact. In addition, a wider range of physicists specialised in this field are now users of the Grid in projects such as EGEE-III or EUFORIA and the future ITER tokamak will will need a Neutral Beam Injector system to heat and feed the plasma and, moreover, to drive toroidal momentum.. The performance of FAFNER-2 on the Grid enhances their results because of the greater number of particles that can be calculated in a reasonable time. It is expected to be run in a sequence with a transport code in a future; first, adapted to the TJ-II stellarator and, later, to different devices -including ITER- by changing the 3D

Three Step Porting

1.- SHMEM to MPI

Tasks:
Replace thread communication method
Optimize code to minimize inter-thread comunication

Performance:
Up to 10% performance gain

2.- MIPS 64 bits to X86 32 bits

Tasks:
Adapt the code to compile on 32 bits architecture
Change of paradigm: shared memory to cluster

Performance:
Only 5% performance gain (same number of slots)
50% more processors can be used

3.- Cluster to Grid

Tasks:
Divide the problem
Create preprocess and post-process wrappers
Submit to Grid employing GridWay

Performance:
Up to 50% performance gain (single node)
High scalability
Wrappers add negligible computing time

Discussion

FAFNER-2 has been succesfully modified to employ MPI instead of SHMEM as a message passing mechanism. Then, it has been ported to Scientific Linux over a X86 machine, allowing it to run on EGEE nodes. It has been proved to be highly scalable, thus is suitable to be executed on the Grid.

After the gridification process, the task of submitting FAFNER-2 to the Grid envolves no difficulty: the final user does not need to change or learn anything. By employing GridWay[2], we can ensure the application is fault-tolerant.

As the results format has not been modified, the application is still suitable to be coupled to a transport code running on a standard computer.

Improvements on the porting process are expected when working with a higher number of particles. Also, a complex grid workflow including FAFNER-2 and ISDEP will be created.

Results

3D plotting of a FAFNER-2 output

Statistic about plasma state generated by FAFNER-2

References

[1]*FAFNER: A Fully 3-D Neutral Beam Injection Code Using Monte Carlo Methods.* Max-Planck-Institut fr Plasmaphysik, 1985
[2]http://www.gridway.org

Laboratorio Nacional de Fusión

Figura A.1: Póster presentado en la conferencia EGEE 08

Bibliografía

[1] http://www.sgi.com/.

[2] http://www.globus.org.

[3] http://public.eu-egee.org.

[4] http://techpubs.sgi.com/library/tpl/cgi-bin/browse.cgi?coll=0650db=man.

[5] http://www.nag.com/.

[6] http://linux.web.cern.ch/linux/scientific3/.

[7] http://www.iter.org/.

[8] *FAFNER: A Fully 3-D Neutral Beam Injection Code Using Monte Carlo Methods*. Max-Planck-Institut fr Plasmaphysik, 1985.

[9] *Globus Toolkit 4: Programming Java Services*. Morgan Kauffmann, 2006.

[10] J. e. a. A.Teubel. Monte carlo simulations of nbi into the tj-ii helical axis stellarator. *Report IPP 4/268*.

[11] M. P. I. Forum. Document for a standard message-passing interface. Technical report, University of Tenneessee, 1994.

[12] I. Foster. What is the grid? a three point checklist. *Grid Today*, 2002.

[13] A. Geist, W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, W. Saphir, T. Skjellum, and M. Snir. Mpi-2: Extending the message-passing interface. *Euro-Par'96*, 1996.

[14] E. Huedo, R. S. Montero, and I. M. Llorente. Evaluating the Reliability of Computational Grids from the End User's Point of View. *Journal of Systems Architecture*, 52(12):727–736, Dec. 2006.

[15] E. Huedo, R. S. Montero, and I. M. Llorente. The gridway framework for adaptative sheduling and execution on grids. *Scalable Computing – Practice and Experience*, 6(3):1–8, 2006.

[16] S. T. I.Foster, C. Kesselman. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):220–222, 2001.

Apéndice B

Autorización de Difusión

El abajo firmante autoriza a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor, la propia memoria.

Fdo: Manuel Rodríguez Pascual